Utilising Machine Learning to Forecast Initial Coin Offering (ICO) Success.

1. Introduction

Crowdfunding is a new way to raise money to start projects or businesses; people or organisations use the internet to collect small amounts of money from many people. An initial coin offering (ICO), a newer crowdfunding method, is widespread and involves people investing by buying new digital money. When they invest through an ICO, they also support the business or project behind the coins. If the project succeeds, the coin's value may rise. Thus, investors may expect the digital coin project to provide a return on their investment, and it would be helpful to have some prediction tools for this. This report will analyse data to predict which ICO projects will be successful or unsuccessful by comparing machine learning models to determine the best one for predicting ICO company success by running the project through CRISP-DM methods.

2. Data Understanding

2.1 Data Structure

The ICO dataset includes 2767 rows and 16 columns.

2.2 Data Types

This dataset includes three different types of data, which are explained in Table 1.

After observing the data type, it could be noticed that some columns of integer type contain binary variables, which are the hasVideo, hasGithub, and hasReddit columns represent "yes" and "no" conditions with numbers of 1 and 0, respectively.

Regarding date, columns ('startDate' and 'endDate') are read as character data that may need to be converted to DateTime to calculate the duration that could be the useful variable.

Type of data	Explanation	Column which stands for that type
Character	Character data that represented in the text	success, brandSlogan, countryRegion, startDate, endDate, platform
integer	The whole numbers without the decimals.	ID, hasVideo, teamSize, hasGithub, hasReddit, minInvestment
number	This type can include both data with whole numbers and have decimal.	Rating, priceUSD, coinNum, distributedPercentage

Table1: Type of ICO data.

2.3 Distribution of Data

The data distribution can be observed in the numerical columns, excluding those containing binary variables. From Figure 1, all these variables represent the right-skewed data meaning there is high extreme value that pulled the tail, which might have affected the prediction. Moreover, these variables contain outliers that can especially be observed in the "teamSize" column. By contrast, the 'rating' variable appears to be likely normally distributed.

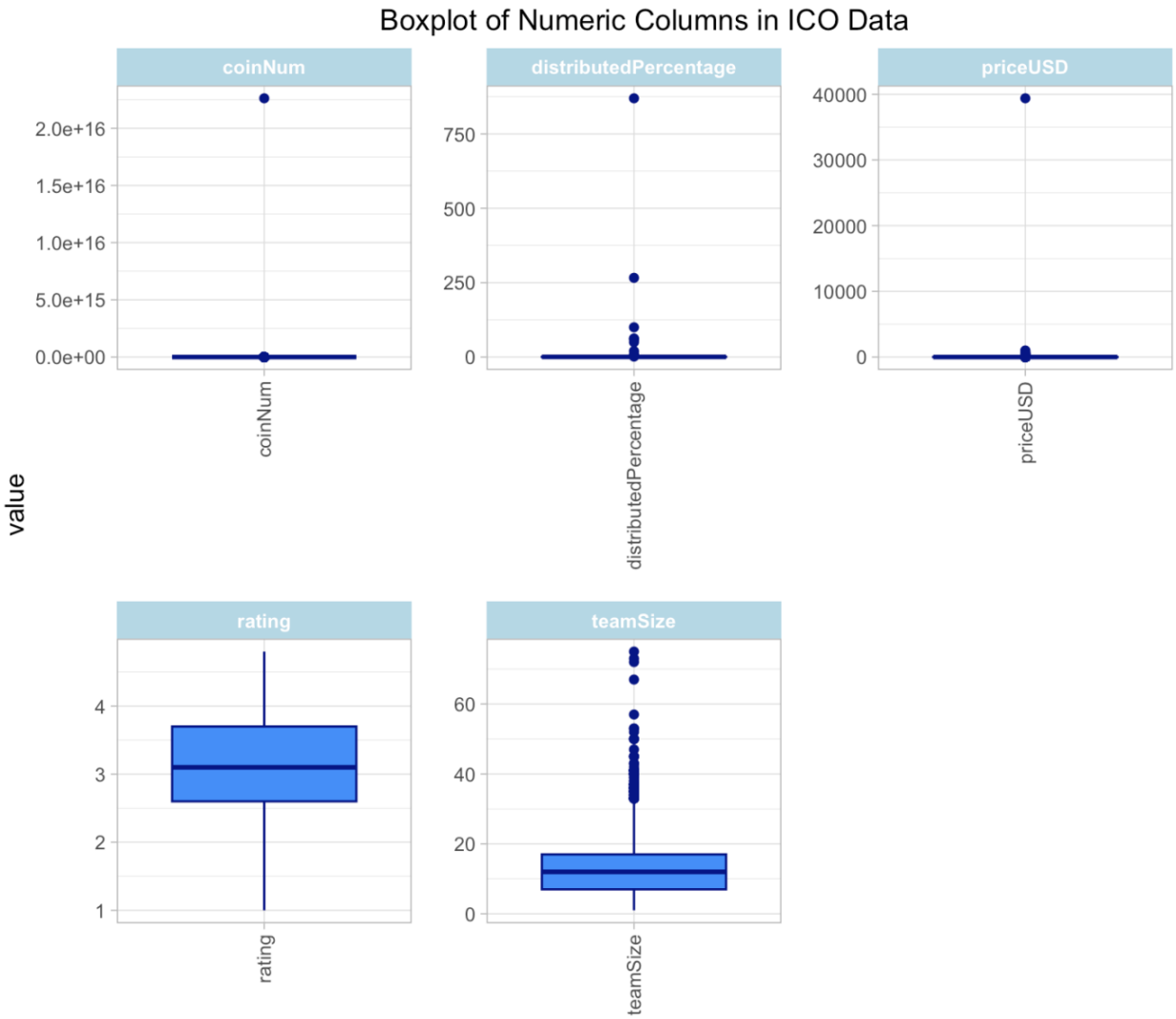

Figure 1: Box plot of Numeric Columns in ICO dataset.

2.4 Understand the Task.

This task focuses on using machine learning to predict the success of the ICO and whether the result should be successful or fail. Thus, the method used in this project is machine learning for classification problems.

3. Data Preparation

3.1 Data Transformation.

3.1.1 Date Transformation.

The columns "startDate" and "endDate" need to change their format from strings to date objects. Then, the range between the end date and start date of ICO will be used as the new duration columns that will be utilised entirely in the analysis.

However, some problems remain, such as the durations containing negative values (for example, -20 and -72) and zero values. It would be impossible if ICO running days were minus and 0, so the wrong value needs to be removed from the dataset. After removing the 52 rows of wrong values, 2715 rows of data are still left.

3.1.2 Categorical Data Transformation.

Changing the target variable (success) into a factor variable before the analysis is helpful in analysis because it can represent the categorical data and be used in machine learning classification.

Variable in “success” columns	Factor variables
Y (yes)	1
N (no)	0

Table 2: The transformation of “success” columns into factor variables.

3.2 Data Selection.

First, remove the unnecessary columns for the prediction, which is ID.

Second, the numerical columns for this dataset will be selected for analysis because most machine learning algorithms require data that can be manipulated in mathematics. Thus, all the numerical columns are selected (hasVideo, teamSize, hasGithub, hasReddit, minInvestment, Rating, priceUSD, coinNum, distributedPercentage and duration), including the target columns (success).

3.3 Relationship of predictors.

Figure 2 below shows the relationship between the numerical columns of the predictors and the predicted value. Darker blue indicates a strong positive correlation, while dark red indicates a strong negative correlation. Most of the variables do not show a strong correlation with each other. This suggests that some variables in this dataset are related to each other, but some of them have weak or no relationship.

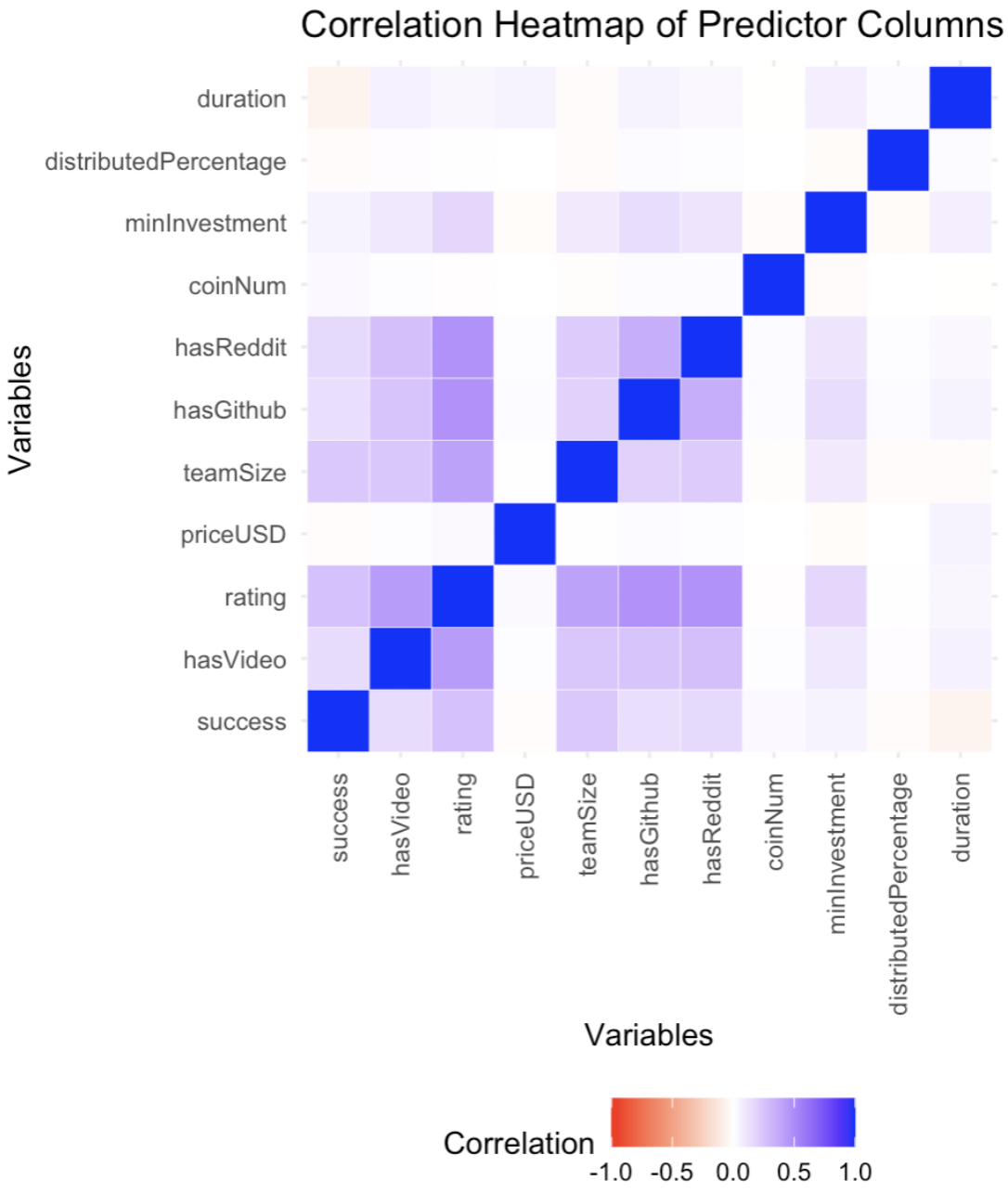

Figure 2: The relationship of numerical values in the ICO dataset.

3.4 Handling the Missing Value.

Columns	Number of missing values
priceUSD	177 values
teamSize	152 values

Table 3: The number of missing values in each column

Handle missing values in the dataset before analysis. Missing data is present in 'priceUSD' and 'teamSize' columns, as shown in Table 3, affecting 320 rows. Due to their weak correlation with the outcome (success) as shown in Figure 2, these rows will be removed, leaving 2,395 rows for further analysis.

3.5 Data Validation.

Data validation is the process of checking the correctness of the ICO dataset.

Figure 1 shows that the dataset contains outliers, as indicated by the box plot of numerical variable distributions. It's important to verify whether these outliers are errors or not, as it can significantly impact the performance of machine learning models.

Figure 3 histograms of binary variables help verify data correctness, confirming that each data point contains only 1 (yes) or 0 (no). Additionally, these histograms indicate the prevalence of each variable's values, such as 'hasGithub', which shows more 'yes' than 'no' responses.

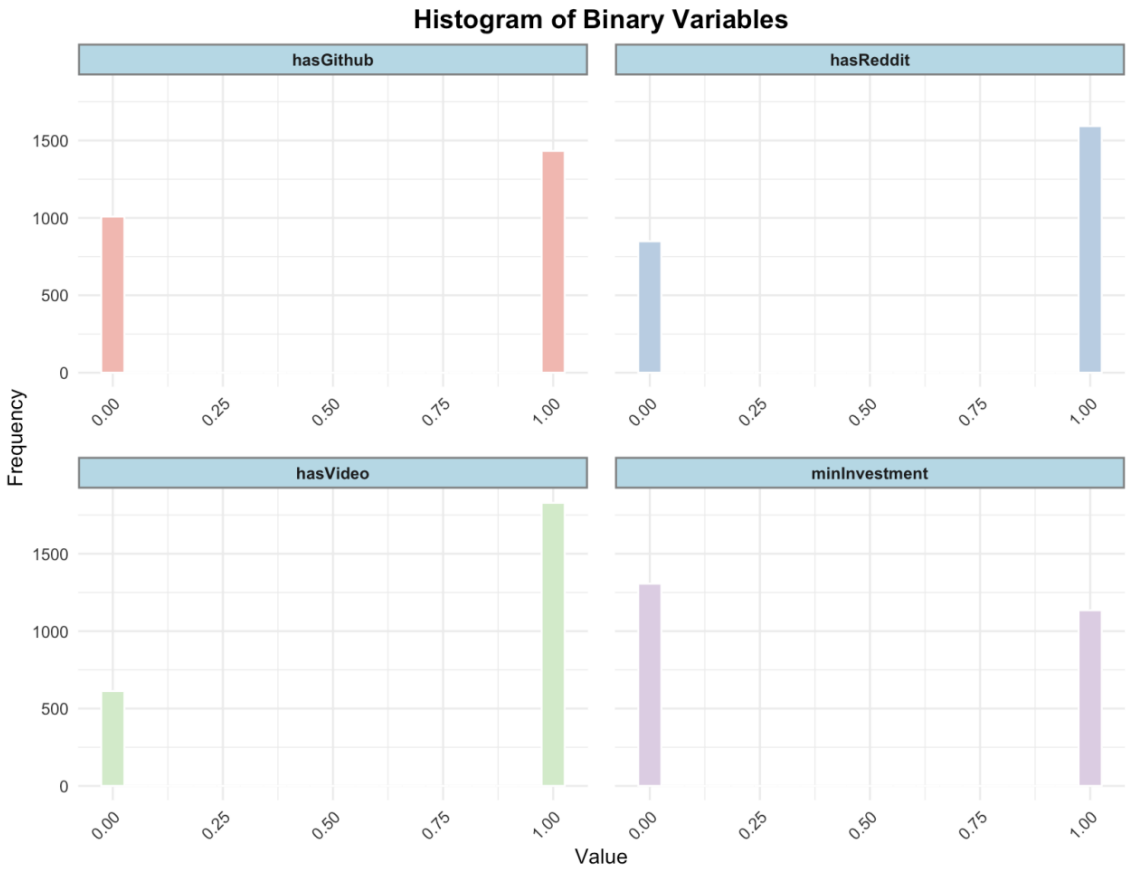

Figure 3: the histogram of binary variables in the ICO dataset.

3.6 Normalisation of data.

Data normalisation rescales numerical values to a 0-1 range, preserving differences in value ratios without altering the scale. This is crucial for comparisons and for scale-sensitive machine learning algorithms like k-nearest neighbors.

$$x_{\ normalised} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

From the equation:

- x is the original value.
- $\min(x)$ is the minimum value of x.
- $\max(x)$ is the maximum value of x.
- $x_{\ normalised}$ is the normalised value.

3.7 Split the Data.

Data is randomly divided into training and test sets. The training set is used to train machine learning models, while the test set evaluates their performance on previously unseen data.

Set of the data	Proportion of splitting
Training set	80 %
Test set	20 %

Table 4: The proportion of splitting

4. Modelling

The machine learning algorithms used for processing the ICO success classification consist of 5 algorithms, as shown in Table 5.

Machine learning algorithms	Known as	Brief explanation
K-Nearest Neighbors	KNN	Assign data to class based on the majority class among the nearest neighbour.
Naive Bayes	NB	Classification data based on the Bayes' theorem and make the Naïve assumption that all features are independent of each other.
Decision tree	DTs	Using the tree-structure to split data by the features for classifying data.
Support Vector Machines	SVMs	Find the best boundary (hyperplane) to separate classes of data.
Artificial Neutral Networks	ANN	Learn the pattern and classify the data by the neutral networks that inspired by human brain.

Table 5: The algorithms processed through the ICO project.

4.1 K-Nearest Neighbors (KNN)

This algorithm is simple to use. However, the algorithm needs the specific value of k (the number of neighbours considered) before it can be processed. The training of the KNN model arguments is explained in Table 6.

KNN modelling	
$knn(train = ico_{train}, test = ico_{test}, cl = ico_{train_{labels}}, k = 47)$	
Arguments	**The input in ICO model**
train	Input ICO data training set.
test	Input ICO data test set.
cl	Factor of true classification of training set setting as the "success" column of training set.
k	Number of the neighbors considered. In this case setting as the $k = \sqrt{total\ rows\ of\ data}$ and setting K as the integer number which could be odd number to reduce the chance of ending with tie vote. After calculation for this ICO dataset k would be setting equal to 47.

Table 6: The input parameter to perform the KNN.

4.2 Naïve Bayes (NB)

This classifier calculates class probabilities for a given input and selects the highest. To address zero-frequency issues in the NB classifier, Laplace smoothing adds a small positive number to frequency counts, ensuring non-zero probabilities. The training of model arguments is explained in Table 7.

Naïve Bayes modelling	
$naiveBayes(x = ico_{train}, y = ico_{train_{labels}};, laplace = 0)$	
Arguments	**The input in ICO model**
x	Input ICO data training set.
y	Input class of ICO training set which is success or fail (1 or 0).
laplace	Setting default as 0 disable the Laplace smoothing which will be change later in the model evaluation section.

Table 7: The input parameter to perform the NB.

4.3 Decision Tree (DT)

The DT training of model arguments is explained in Table 8.

Decision tree modelling	
$C5.0(x = ico_{train}, y = ico_{train_{labels}}, trails = 1)$	
Arguments	**The input in ICO model**
x	Input ICO data training set.
y	Input class of ICO training set which is success or fail (1 or 0).
trials	Specific the number of boosting iterations to improve the performance of model by focusing on the data that its previous learners have misclassified: setting as default as 1.

Table 8: The input parameter to perform the DT.

Table 9 displays the importance scores of features for constructing a decision tree. It highlights that only four features are significant, while others score 0, indicating their lack of importance. The ICO rating, with the highest score of 100, emerges as the most significant predictor of the outcome variable.

Features	Important Score
Rating	100.00
Team size	54.38
Duration of project	36.59
Have Reddit	21.87

Table 9: The important score of ICO feature to building the decision tree.

The ICO data was used to build the decision tree, as shown in Figure 4. It starts the root node with ‘rating’ and splits the data at each node, and ends the prediction at the leaf node, which shows the final decision and the class distribution.

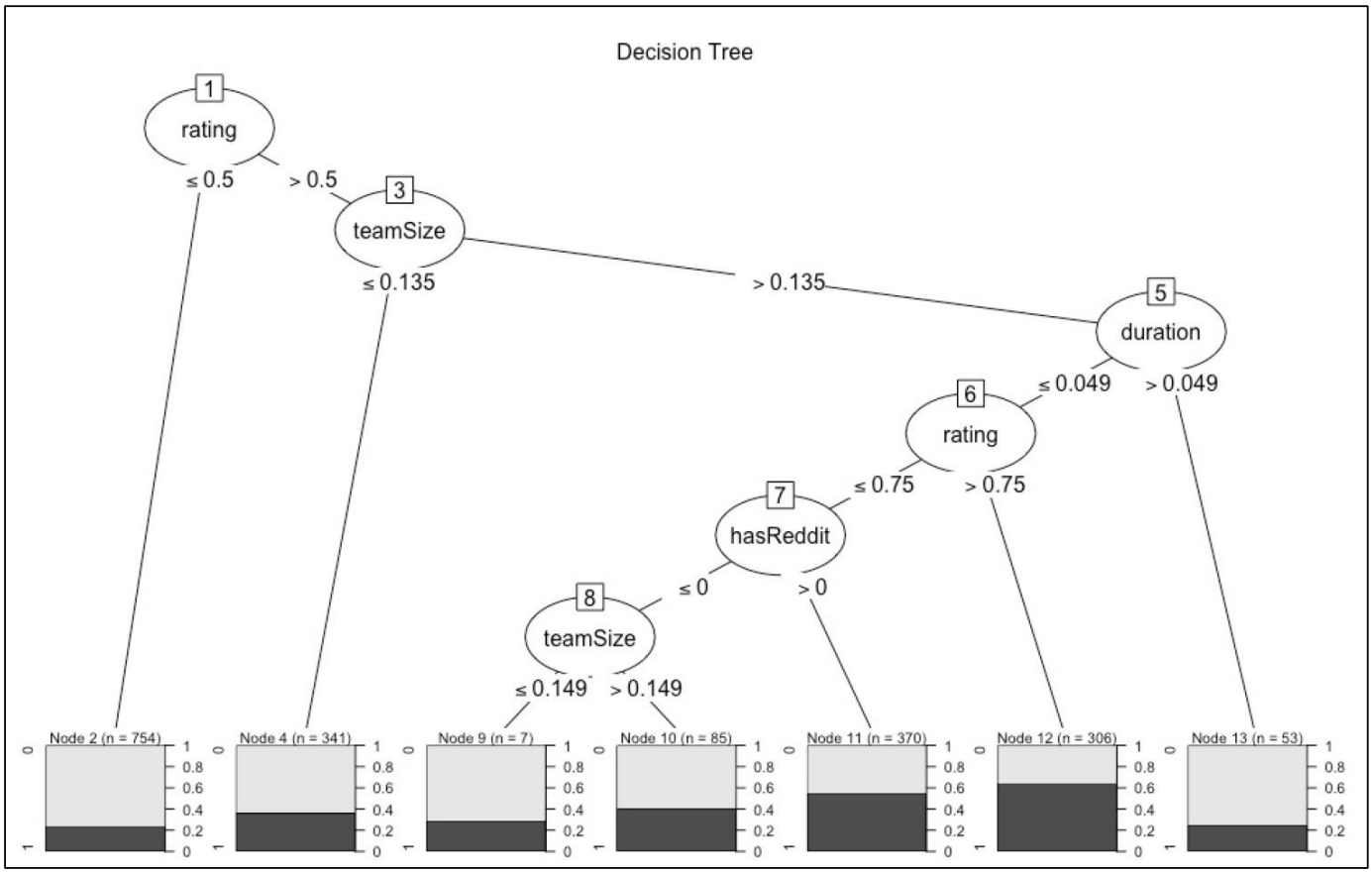

Figure 4: The decision tree of ICO.

4.4 Support Vector Method (SVM)

The training of the SVM model arguments is explained in Table 10.

SVM modelling	
$ksvm(x = success \sim . , data = ico_{train}, kernel = "vanilladot")$	
Arguments	**The input in ICO model**
x	Specifies all numerical variables in ICO data as predictors for 'success'.
data	Input ICO data training set.
kernel	The kernels can transform the input data space in different way to make it easy of hyperplane to separate the classes. In this model setting as 'vanilladot' means a linear kernel.

Table 10: The input parameter to perform the SVM.

4.5 Artificial Neutral Network (ANN)

The training of the ANN model arguments is explained in Table 11.

ANN modelling	
$neuralnet(formula = success \sim . , data = ico_{train}, hidden = 1)$	
Arguments	**The input in ICO model**
formula	The model will predict the "success" features based on all other features.
data	Input ICO data training set.
hidden	The number of hidden neurons in each layer setting as 1 hidden neuron.

Table 11: The input parameter to perform the ANN.

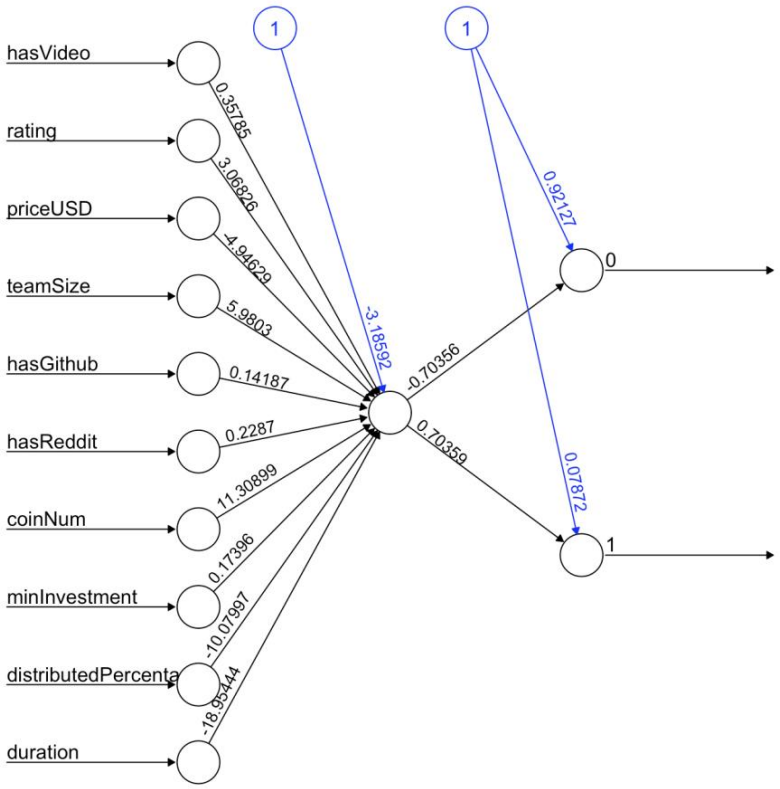

Figure 5: Artificial Neural Network Structure for ICO Success Prediction.

Figure 5 shows the model of ANN that the inputs are connected to the hidden layer, which is set as 1 in this case, and the neurons are connected to the two output nodes that predict the class of ICO that is successful or fails.

5. Evaluation

In terms of the evaluation step, this method will focus on improving the performance of each model by adjusting the parameters compared to the results to find which setting of the model argument might be the best. The process of evaluating each classification model is evaluated through the two methods that are:

1. The confusion metric
2. Receiver Operating Characteristics (ROC) and Area under the curve (AUC)

Confusion metric.

In this model, the evaluation uses four important metrics that are:

- Accuracy: Measures overall correctness of the model
- Precision: Measures correctness of positive predictions
- Recall (Sensitivity): Assess the model's ability to identify all actual positives.
- F-1 Score: This score considers both precision and recall providing a balanced evaluation.

ROC and AUC

The ROC Curve is used to illustrate how well that model separates successful and unsuccessful results. Moreover, the AUC is used to measure the model's accuracy, and a higher value will indicate better performance.

5.1 Evaluation of KNN Models

Adjusting the value of K can improve the performance of the KNN model. At the start of the model, k was set to equal 47 and trying to adjust K as an odd number to reduce the chance of ending with a tie vote. The results after adjustment are shown in Figure 6 and Table 12.

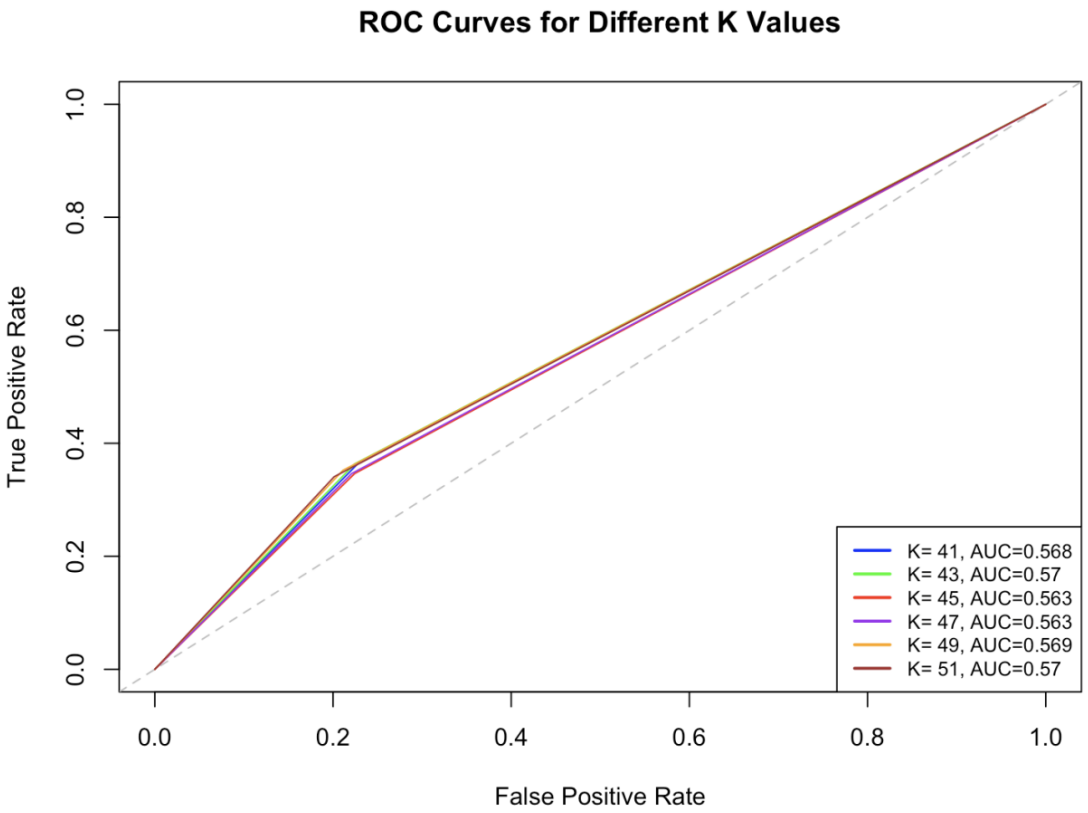

Figure 6: The ROC curves of comparing KNN models.

The ROC curves in Figure 6 do suggest a similar performance between different K values because the lines are very close together in the different K models. Thus, it would focus on evaluating metrics to evaluate performance.

Adjusting Parameter.	**Evaluate metrics**			
	accuracy	**precision**	**recall**	**F-1 score**
K = 41	0.622	0.364	0.481	0.414
K = 43	0.624	0.364	0.485	0.415
K = 45	0.620	0.347	0.476	0.396
K = 47	0.618	0.340	0.472	0.396
K = 49	0.628	0.352	0.492	0.410
K = 51	0.628	0.352	0.492	0.410

Table 12: Performance metrics for KNN models.

Based on the result from Table 12, the model with K = 49 has the highest accuracy, recall, and F-1 score. The model with K = 51 has all values equal to K = 49, which might show that increasing the K value does not improve the model anymore. Thus, the model with K = 49 might be the best model in this trial adjustment list.

5.2 Evaluation of Naïve Bayes Models

Using Laplace smoothing, a technique for handling zero probability in datasets, could improve the performance of the Naive Bayes.

Adjusting Parameter.	**Evaluate metrics**			
	accuracy	**precision**	**recall**	**F-1 score**
Laplace = 0	0.631	0.040	0.467	0.073
Laplace = 1	0.631	0.040	0.467	0.073

Table 13: Performance metrics for NB models.

Table 13 illustrates that both models with Laplace smoothing (Laplace = 1) or without doing all evaluated metrics are still equal with support by the curve and value of AUC in Figure 7, which means the performance of Laplace smoothing might not improve the performance.

This model has a low precision value, implying that it does not perform well in correctly identifying the success case, which also leads to a low F-1 score. Thus, this model does not seem to be the appropriate model for predicting ICO success.

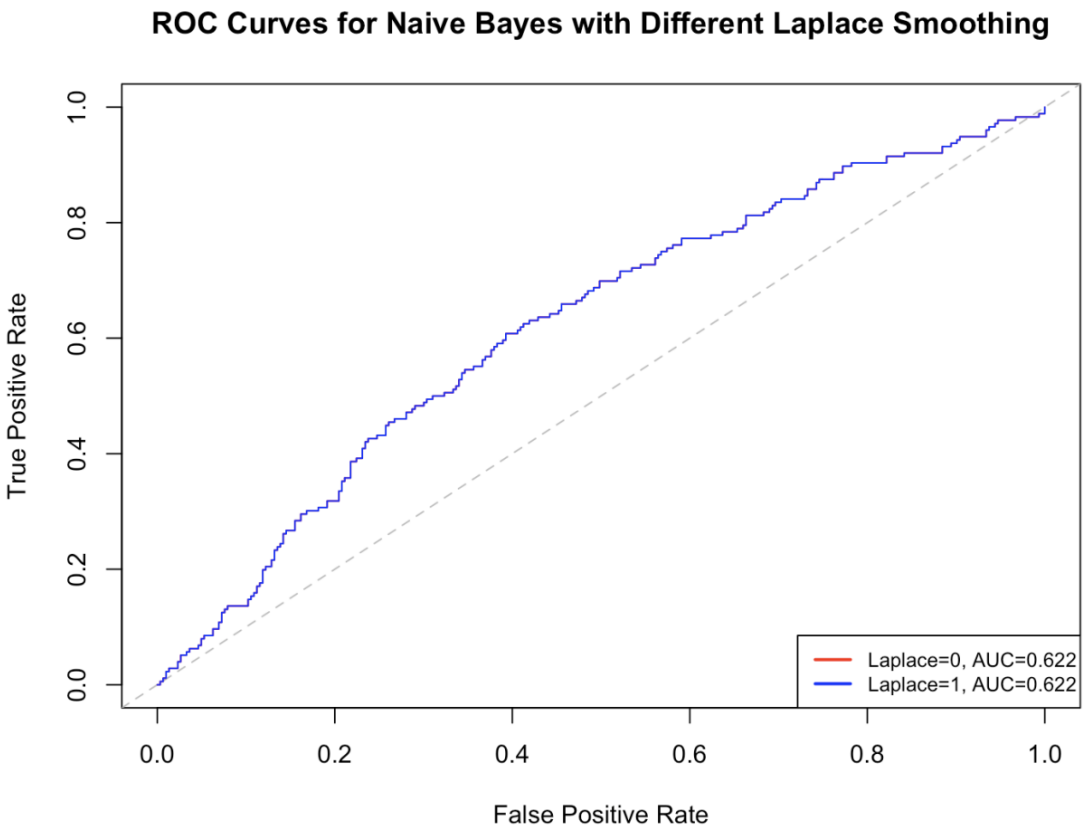

Figure 7: The ROC curves of comparing NB models.

5.3 Evaluation of Decision Tree Models

Adjusting the number of boosting iterations could improve the decision tree model's performance. This technique improves accuracy by combining several weak models into a stronger classifier.

Adjusting Parameter.	Evaluate metrics			
	accuracy	precision	recall	F-1 score
Trails = 1	0.639	0.381	0.511	0.436
Trails = 2	0.639	0.381	0.511	0.436
Trails = 3	0.635	0.369	0.504	0.426
Trails = 4	0.645	0.386	0.523	0.444
Trails = 5	0.647	0.483	0.521	0.501
Trails = 6	0.647	0.483	0.521	0.501

Table 14: Performance metrics for DT models.

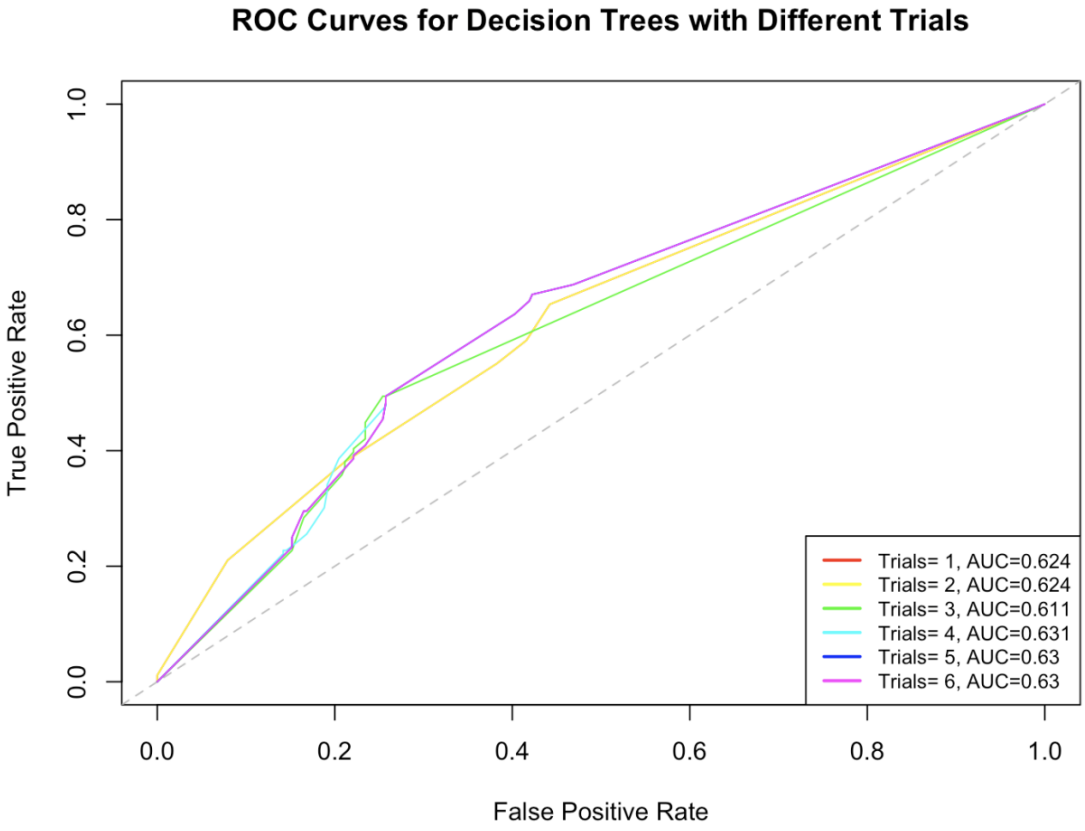

Figure 8: The ROC curves of comparing DT models.

From Table 14, with an increase in the number of boosting trials, the model provided an improvement in accuracy and precision, suggesting that the model is more accurate in predicting the positive result (success). However, the model performances seem not to

increase after passing the 5 trials supported by the ROC curves in Figure 8, which 5 trials might be the highest evaluation metric score of this model.

After that, try to evaluate the results of 5 trials by this time, selecting only 4 important variables: Rating, Team size, Duration of the project, and Have Reddit**,** as shown in Table 9. However, the evaluation results, as Table 15 suggests the performance is reduced after cutting some variables off.

Adjusting Parameter.	**Evaluate metrics**			
	accuracy	**precision**	**recall**	**F-1 score**
Trails = 5	0.647	0.483	0.521	0.501
Trails = 5 Selected only 4 important variables	0.633	0.278	0.500	0.358

Table 15: Performance metrics for DT models.

5.4 Evaluation of Support Vector Models

Changing the kernel function, which transforms data into a higher-dimensional space to adjust the way to find the hyperplane to separate the class of data, could improve the efficiency of the SVM models. This ICO model predicts improvement might progress through four different kernels; the results are provided in Table 16 and Figure 9.

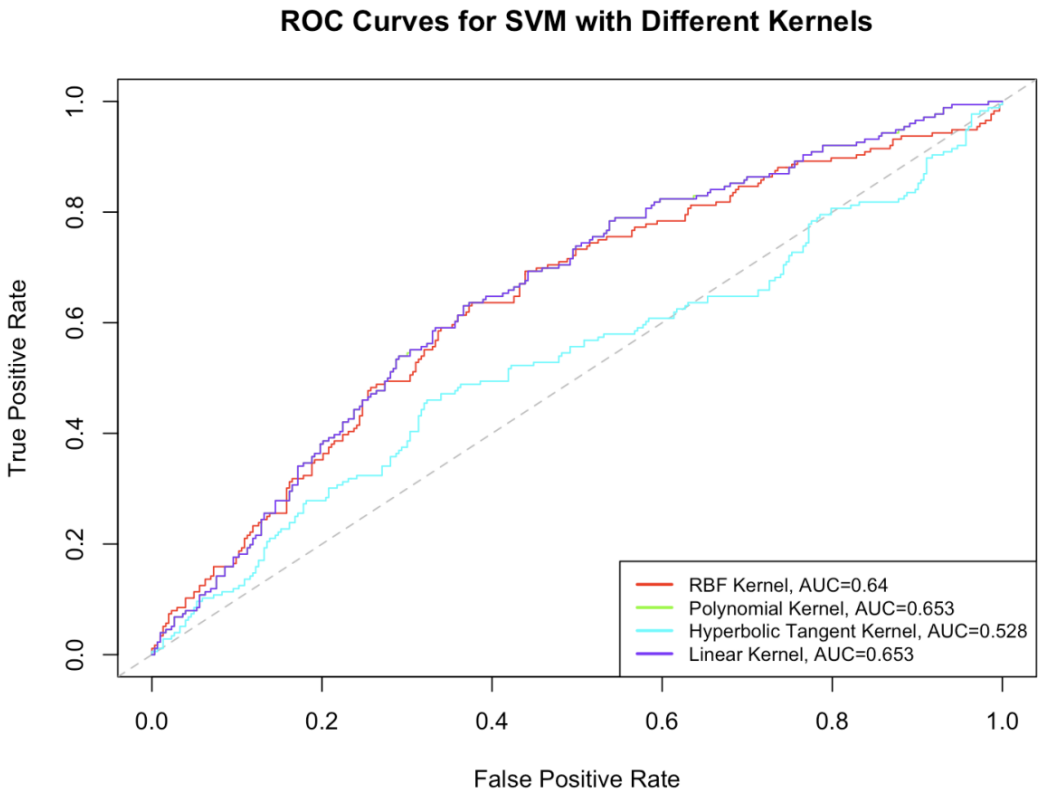

Figure 9: The ROC curves of comparing SVM models.

Adjusting Parameter.	Evaluate metrics			
	accuracy	precision	recall	F-1 score
Kernel = 'vanilladot' (linear kernel)	0.641	0.392	0.515	0.445
Kernel = 'rbfdot' (radial basis function)	0.633	0.324	0.500	0.393
Kernel = 'polydot' (polynomial kernel)	0.641	0.392	0.515	0.445
Kernel = 'tanhdot' (hyperbolic tangent kernel)	0.595	0.449	0.449	0.449

Table 16: Performance metrics for SVM models.

The results from Table 16 show that the linear and polynomial kernels have the highest accuracy and F-1 score, suggesting that these two functions could perform better among the four, supporting the result by ROC curves in Figure 9.

5.5 Evaluation of Artificial Neutral Network Model

Adjusting the number of hidden layers might be the best option to increase the ANN model's performance. Test through the trial and error of adjusting hidden layers as in Table 17.

Adjusting Parameter.	Evaluate metrics			
	accuracy	**precision**	**recall**	**F-1 score**
0 Hidden layers	0.651	0.398	0.534	0.456
1 Hidden layer with 1 neuron	0.639	0.415	0.510	0.458
1 Hidden layer with 2 neurons	0.637	0.398	0.507	0.446
1 Hidden layer with 3 neurons	0.641	0.426	0.514	0.466
1 Hidden layer with 4 neurons	0.631	0.381	0.496	0.431
2 Hidden layers with 1 neuron first and 3 neurons second	0.639	0.415	0.510	0.458
2 Hidden layers with 1 neuron first and 4 neurons second	0.639	0.415	0.510	0.458

Table 17: Performance metrics for ANN models.

The result from Table 17 suggests that the ANN model with no hidden layers performs best in terms of accuracy and recall. However, the 1 hidden layer model with 3 neurons shows the highest F-1 score and some improvement in precision, which means this model is good for predicting a positive result(success).

Figure 10 suggests evidence to support the fact that the AUC values of o layers and 1 layer between 1 and 3 neurons have a high value. However, the performance of the 1 layer with 4 neurons for prediction seems to drop.

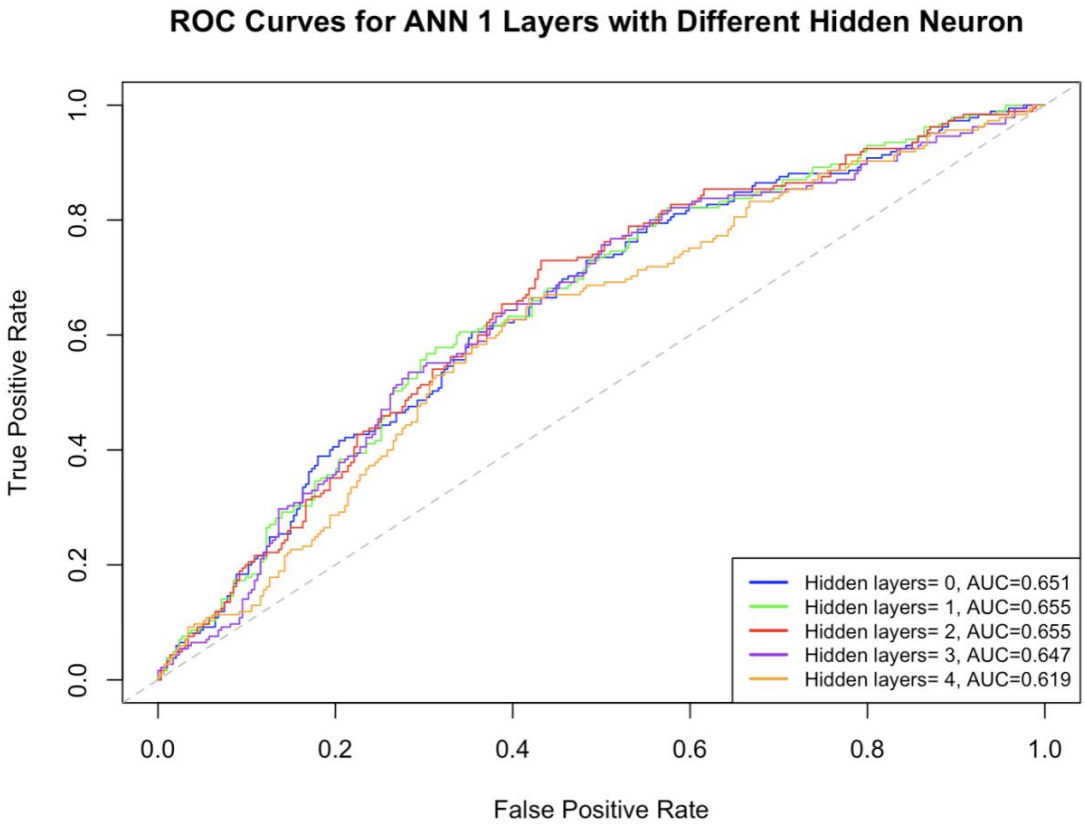

Figure 10: The ROC curves of comparing ANN models.

Thus, this ICO prediction model focuses on predicting with a balance of precision and recall to minimise the case that the model incorrectly predicts when it is actually unsuccessful, which the selection of a model with three hidden layers might be the best based on ROC curves and evaluation metrics.

5.6 Comparing the model performance.

Pick the model with different algorithms that have the best performance and compare them to find which model is the most appropriate to use to predict the success of an ICO.

Machine Learning algorithms	Evaluate metrics			
	accuracy	precision	recall	F-1 score
KNN which K = 49	0.618	0.340	0.472	0.396
NB with Laplace smoothing	0.631	0.040	0.467	0.073
DTs with 5 boosting iterations	0.647	0.483	0.521	0.501
SVM with linear and polynomial kernel	0.641	0.392	0.515	0.445
ANN with 1 Hidden layer with 3 neurons	0.641	0.426	0.514	0.466

Table 18: Comparison of Machine Learning Algorithm Performance.

Table 18 illustrates that the decision tree model with boosting iteration provides the highest accuracy and the F-1 score, which indicates a good balance between precision and recall, which might be the best model that has a balance to predict the ICO's success. The ANN and SVM also perform well, but all the metrics are slightly lower than those of the DT. KNN have a low evaluation metric score compared to DT, ANN and SVM. The Naïve Bayes model has a very low F-1 score, which might suggest that the NB models are not appropriate for predicting ICO success.

Thus, based on the balance of all metrics, the DT model seems to be the best model for predicting the success of an ICO, which has the highest ability to cover the true positive (success) result while minimising the rate of false positive that might cause the problem of misunderstanding or wrong decision in the future.

6. Conclusion

In conclusion, the decision tree with five boosting iterations might be the most appropriate machine learning model to predict the success of the ICO. This model is a robust and well-balanced prediction based on the evidence of the highest evaluation metrics. While the ANN and SVM show competitive performance, their metrics are slightly lower than those of the DTs model.

However, this study also faces limitations, notably the model's overall low accuracy of approximately 60%, which might be the effect of some factors, for example, the data distribution with the right skew and some potential outliers that could impact the prediction result.

For future studies, exploring the data pre-processing techniques, experimenting with a broader range of adjustments to the model parameters, and trying more complex models like deep learning could provide access to deeper insights and improve predictive performance, which might be useful for the ICO’s successful prediction in the future.

7. Appendix

7.1 Data observation and preprocess

```
# ICO datasets
# install the necessary packages
install.packages("gmodels")
library(gmodels)
library(dplyr)
library(ggplot2)
library(tidyr)
library(class)
library(tidyverse)
library(ROCR)
library(caret)
#-----------------------------------------------------------------------------------------------
# 1. Exploring, preparing and transforming the data
# read data
# using read csv
ico <- read.csv("LUBS5990M_courseworkData_2324.csv", stringsAsFactors = FALSE, encoding = 'UTF-8')

# checking the data
str(ico)
# priceUSD has 180 and teamSize has 154
summary(ico)

# remove the variable that useless for prediction
# remove id
ico_new <- ico[-1]

# Create box plot to see the distribution of nummeric variable
ico_new %>%
  select(coinNum, distributedPercentage, priceUSD, rating, teamSize) %>%  # Select specific columns
  pivot_longer(cols = everything()) %>%
  ggplot(aes(x = name, y = value)) +
  geom_boxplot(fill = "dodgerblue", colour = "darkblue") +  # Customize boxplot appearance
  facet_wrap(~name, scales = "free") +
  theme_light() +  # A lighter theme
  theme(
    axis.text.x = element_text(angle = 90, vjust = 0.5),
    axis.title.x = element_blank(),  # Remove x-axis title
    axis.title.y = element_text(size = 12),
    strip.background = element_rect(fill = "lightblue"),  # Color behind facet labels
    strip.text = element_text(face = "bold"),  # Bold facet labels
    panel.spacing = unit(1, "lines"),  # Spacing between facets
    plot.title = element_text(hjust = 0.5)  # Center the plot title
  ) +
  labs(title = "Boxplot of Numeric Columns in ICO Data")
```

```
# Create histrogram plot to see the distribution of binary variable
ico_new %>%
  select(hasGithub, hasReddit, hasVideo, minInvestment) %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(x = value, fill = name)) +
  geom_histogram(bins = 20, color = "white") +
  scale_fill_brewer(palette = "Pastel1") + # Use a more subtle color palette
  facet_wrap(~name, scales = "free_x", ncol = 2) +  # Organize plots in 2 columns
  theme_minimal() +
  theme(
    strip.background = element_rect(fill = "lightblue", colour = "grey50", size = 1),
    strip.text = element_text(face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, color = "grey20"),
    axis.text.y = element_text(color = "grey20"),
    legend.position = "none", # Hide the legend if not necessary
    panel.spacing = unit(1, "lines"), # Adjust spacing between facets
    plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
    plot.subtitle = element_text(hjust = 0.5, size = 10)
  ) +
  labs(
    title = "Histogram of Binary Variables",
    x = "Value",
    y = "Frequency"
  )
#-----------------------------------------------------------------------------------------------
# Date transformation
ico_new$startDate <- as.Date(ico_new$startDate, format="%d/%m/%Y")
ico_new$endDate <- as.Date(ico_new$endDate, format="%d/%m/%Y")
# calculate the duration in days as a new column
ico_new$duration <- as.integer(ico_new$endDate - ico_new$startDate)

# remove the wrong duration value
ico_new <- ico_new %>%
  filter(duration > 0)

# changing the success columns in to binary variable
ico_new$success <- as.factor(ifelse(ico_new$success == 'Y', 1, 0))

# Check correlation between numerical variables(predictors)
ico_new_corr <- ico_new
ico_new_corr$success <- as.numeric(ico_new_corr$success)

# Calculate correlation matrix
# use complete.obs to handle any NA values

numerical_cols <- ico_new_corr[sapply(ico_new_corr, is.numeric)]
# Calculate correlation matrix
correlation_matrix <- cor(numerical_cols, use = "complete.obs")

# Melt the correlation matrix for plotting
correlation_melted <- melt(correlation_matrix)

# Plot heatmap
ggplot(data = correlation_melted, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 0, limit = c(-1,1), space = "Lab", name="Correlation") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        axis.text.y = element_text(angle = 0, vjust = 0.5, hjust=1),
        legend.position = "bottom") +
  coord_fixed() +
  labs(title = "Correlation Heatmap of Predictor Columns",
       x = "Variables",
       y = "Variables")
```

```
# priceUSD has 177 and teamSize has 152
summary(ico_new)

# check total row with missing value
total_rows_with_missing_values <- sum(!complete.cases(ico))
# Filtering the columns with have missing value out
ico_new <- ico_new %>% filter(!if_any(everything(), is.na))

# check the missing value again
summary(ico_new)

#-----------------------------------------------------------------------------------------------------
# Normalizing numeric values ---> data preprocessing
normalize <- function(x) {
  return ((x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE)))
}

# Using sapply to identify numeric columns and then normalising
ico_numeric_cols <- sapply(ico_new, is.numeric)
# Apply normalization to numeric columns
ico_new_normalized <- ico_new
ico_new_normalized[ico_numeric_cols] <- as.data.frame(lapply(ico_new[ico_numeric_cols], normalize))
# range from 0-1
summary(ico_new_normalized)
str(ico_new_normalized)

# select only numerical col and keep target variable
ico_new_normalized_num <- select(ico_new_normalized, -c(brandSlogan, countryRegion, startDate, endDate, platform))

#-----------------------------------------------------------------------------------------------------
```

7.2 KNN modelling

```
#-----------------------------------------------------------------------------------------------
# 2 Trainning the model

# Now split into training (80 percent),
# and test data (20 percent)
ico_size <- floor(0.80 * nrow(ico_new_normalized_num))
# set seed
set.seed(12345)
# split
trainning <- sample(nrow(ico_new_normalized_num), ico_size)
ico_train <- select(ico_new_normalized_num[trainning, ], -success)
ico_test <- select(ico_new_normalized_num[-trainning, ], -success)

#class selected only first col = "success"
ico_train_labels <- ico_new_normalized_num[trainning, 1]
ico_test_labels <- ico_new_normalized_num[-trainning, 1]

# K-NN classification
# specific the number of K
K = floor(sqrt(nrow(ico_new_normalized_num)))

#  Using an odd number of K will reduce the chance of ending with a tie vote.
if (K %% 2 == 0) { # If K is even
  K = K - 1
}

ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=K)

#-----------------------------------------------------------------------------------------------
#3. Evaluating model performance
install.packages("caret", dependencies = TRUE)
library('caret')

CrossTable(x = ico_test_labels, y = ico_test_pred, prop.chisq=FALSE)

ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=K-6) #K=41
ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=K-4) #K=43
ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=K+2) #K=47
ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=K+2) #K=51

confusionMatrix(ico_test_labels, ico_test_pred, positive = "1")
prec = posPredValue(ico_test_labels, ico_test_pred, positive = "1") # this is precision
rec =sensitivity(ico_test_labels, ico_test_pred, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
```

```
#-----------------------------------------------------------------------------------------------
# ROC and AUC
# setting K values and colours
K_values <- c(41, 43, 45, 47, 49, 51)
K_names <- paste("K=", K_values)
colors <- c("blue", "green", "red", "purple", "orange", "brown")

# Create list to store predictions
predictions_list <- list()

# Predictions for each K
for (i in 1:length(K_values)) {
  predictions_list[[i]] <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k = K_values[i])
}

# create plot
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
Different K Values")
abline(a = 0, b = 1, lty = 2, col = "gray")

# create vector to store AUC values
AUC_values <- numeric(length(predictions_list))

# Loop to calculate and plot each ROC curve
for (i in 1:length(predictions_list)) {
  # Convert to numeric
  test_pred_numeric <- as.numeric(as.character(predictions_list[[i]]))
  test_labels_numeric <- as.numeric(as.character(ico_test_labels))

  # Create prediction object for ROC curve
  pred_object <- prediction(test_pred_numeric, test_labels_numeric)
  # Calculate the performance of the predictor
  perf <- performance(pred_object, "tpr", "fpr")
  # Calculate AUC
  auc <- performance(pred_object, "auc")
  AUC_values[i] <- auc@y.values[[1]]
  # Plot ROC curve
  plot(perf, add = TRUE, col = colors[i])
}
# legend to include AUC values
K_names_with_auc <- paste(K_names, ", AUC=", round(AUC_values, 3), sep="")
# Add legend
legend("bottomright", K_names_with_auc, col = colors, lwd = 2, cex = 0.8)
#-----------------------------------------------------------------------------------------------
```

7.3 NB modelling

```
# VSM
# step1: install necessary package
# install the necessary package

install.packages('e1071')
library(e1071)

#-----------------------------------------------------------------------------------------------
# 2 Trainning the model
# the data into test and train same as before
# Now split into training (80 percent),
# and test data (20 percent)
ico_size <- floor(0.80 * nrow(ico_new_normalized_num))
# set seed
set.seed(12345)
# split
trainning <- sample(nrow(ico_new_normalized_num), ico_size)
ico_train_nv <- ico_new_normalized_num[trainning, ]
ico_test_nv <- ico_new_normalized_num[-trainning, ]

#class selected only first col = "success"
ico_train_nv_labels <- ico_new_normalized_num[trainning, 1]
ico_test_nv_labels <- ico_new_normalized_num[-trainning, 1]

# train the model
ico_classifier_nv <- naiveBayes(ico_train_nv, ico_train_nv_labels, laplace = 0)

# Test the model
ico_nv_prediction <- predict(ico_classifier_nv, select(ico_test_nv, -success))
#-----------------------------------------------------------------------------------------------
# Step 3: Evaluating model performance

CrossTable(x = ico_test_nv_labels, y = ico_nv_prediction, prop.chisq=FALSE)

confusionMatrix(ico_test_nv_labels, ico_nv_prediction, positive = "1")
prec = posPredValue(ico_test_nv_labels, ico_nv_prediction, positive = "1") # this is precision
rec =sensitivity(ico_test_nv_labels, ico_nv_prediction, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
#-----------------------------------------------------------------------------------------------
# Improve the model

# Improve the model performance by doing Laplace smoothing.
ico_classifier_nv <- naiveBayes(ico_train_nv, ico_train_nv_labels, laplace = 1)
ico_nv_prediction <- predict(ico_classifier_nv, select(ico_test_nv, -success))

CrossTable(x = ico_test_nv_labels, y = ico_nv_prediction, prop.chisq=FALSE)

confusionMatrix(ico_test_nv_labels, ico_nv_prediction, positive = "1")
prec = posPredValue(ico_test_nv_labels, ico_nv_prediction, positive = "1") # this is precision
rec =sensitivity(ico_test_nv_labels, ico_nv_prediction, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
#-----------------------------------------------------------------------------------------------
```

```
# ROC and AUC curve

# Naive Bayes models with different Laplace smoothing
ico_classifier_nv0 <- naiveBayes(ico_train_nv, ico_train_nv_labels, laplace = 0)
ico_classifier_nv1 <- naiveBayes(ico_train_nv, ico_train_nv_labels, laplace = 1)
# Predict probabilities
prob_predictions_nv0 <- predict(ico_classifier_nv0, select(ico_test_nv, -success), type = "raw")
prob_predictions_nv1 <- predict(ico_classifier_nv1, select(ico_test_nv, -success), type = "raw")

# the positive class is the second column
test_pred_numeric0 <- prob_predictions_nv0[, 2]
test_pred_numeric1 <- prob_predictions_nv1[, 2]
# change to numeric test labels
test_labels_numeric <- as.numeric(as.character(ico_test_nv_labels))

# setting the plot
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
Naive Bayes with Different Laplace Smoothing")
abline(a = 0, b = 1, lty = 2, col = "gray")

# Set up legend and colors
classifier_names <- c("Laplace=0", "Laplace=1")
colors <- c("red", "blue")

# Setting vector to store AUC values
AUC_values <- numeric(2)

# List of prediction probabilities for looping
predictions_list <- list(test_pred_numeric0, test_pred_numeric1)

# Loop
for (i in 1:2) {
  # Create prediction object for ROC curve
  pred_object <- prediction(predictions_list[[i]], test_labels_numeric)
  # Calculate the performance
  perf <- performance(pred_object, "tpr", "fpr")
  # Calculate AUC
  auc <- performance(pred_object, "auc")
  AUC_values[i] <- auc@y.values[[1]]
  # Plot ROC curve
  plot(perf, add = TRUE, col = colors[i])
}

# Update legend to include AUC values
classifier_names_with_auc <- paste(classifier_names, ", AUC=", round(AUC_values, 3), sep="")
# Add legend
legend("bottomright", classifier_names_with_auc, col = colors, lwd = 2, cex = 0.8)
#-----------------------------------------------------------------------------------------------
```

7.4 DT modelling

```
# Decision Trees
# step1: install necessary package
# install the necessary package
install.packages("C50")
library(C50)


#-------------------------------------------------------------------------------------------------
# 2 Trainning the model

# the data into test and train same as before
# Now split into training (80 percent),
# and test data (20 percent)
ico_size <- floor(0.80 * nrow(ico_new_normalized_num))
# set seed
set.seed(12345)
# split
trainning <- sample(nrow(ico_new_normalized_num), ico_size)
ico_train_dt <- ico_new_normalized_num[trainning, ]
ico_test_dt <- ico_new_normalized_num[-trainning, ]

#class selected only first col = "success"
ico_train_dt_labels <- ico_new_normalized_num[trainning, 1]
ico_test_dt_labels <- ico_new_normalized_num[-trainning, 1]

dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 1)
dt_model

summary(dt_model)

#-------------------------------------------------------------------------------------------------
library(partykit)

# Assuming 'dt_model' is your C5.0 model
dt_party <- as.party(dt_model)

# Plot the decision tree with customized font size
plot(dt_party,
     main = "Decision Tree",
     tp_args = list(gp = gpar(fontsize = 8)),
     gp_args = list(gp = gpar(fontsize = 8)))
png("decision_tree.png", width = 800, height = 800)
#-------------------------------------------------------------------------------------------------

C5imp(dt_model)

dt_pred <- predict(dt_model, ico_test_dt)

#-------------------------------------------------------------------------------------------------
#3. Evaluating model performance

CrossTable(x = ico_test_dt_labels, y = dt_pred, prop.chisq=FALSE)
confusionMatrix(ico_test_dt_labels, dt_pred, positive = "1")
prec = posPredValue(ico_test_dt_labels, dt_pred, positive = "1") # this is precision
rec =sensitivity(ico_test_dt_labels, dt_pred, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
#-------------------------------------------------------------------------------------------------
```

```
# Improve the performance by changing the number of boosting iterations.
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 1)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 2)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 3)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 4)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 5)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 6)
dt_model <- C5.0(x = select(ico_train_dt, -c(success, hasVideo, priceUSD, hasGithub, coinNum, minInvestment,
distributedPercentage)), y = ico_train_dt$success, trials = 5)
dt_pred <- predict(dt_model, ico_test_dt)

CrossTable(x = ico_test_dt_labels, y = dt_pred, prop.chisq=FALSE)
confusionMatrix(ico_test_dt_labels, dt_pred, positive = "1")
prec = posPredValue(ico_test_dt_labels, dt_pred, positive = "1") # this is precision
rec =sensitivity(ico_test_dt_labels, dt_pred, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1

#-----------------------------------------------------------------------------------------------------
# ROC and AUC

# Create a list of decision tree models with different numbers of trials
trial_values <- 1:6
dt_models <- list()
for (i in trial_values) {
  dt_models[[i]] <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = i)
}

# Predict probabilities
predictions_list <- lapply(dt_models, function(model) {
  predict(model, ico_test_dt, type = "prob")[, "1"]
})

# Change to numeric
test_labels_numeric <- as.numeric(ico_test_dt$success)

# create plot
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
Decision Trees with Different Trials")
abline(a = 0, b = 1, lty = 2, col = "gray")
# Names for legend and colors use the rainbow col
colors <- rainbow(length(trial_values))
trial_names <- paste("Trials=", trial_values)

# create vector to store AUC values
AUC_values <- numeric(length(trial_values))

# Loop through each set of predictions
for (i in seq_along(predictions_list)) {
  # Create prediction object for ROC curve
  pred_object <- prediction(predictions_list[[i]], test_labels_numeric)
  # Calculate the performance
  perf <- performance(pred_object, "tpr", "fpr")
  # Calculate AUC
  auc <- performance(pred_object, "auc")
  AUC_values[i] <- auc@y.values[[1]]  # Store AUC value
  # Plot ROC curve
  plot(perf, add = TRUE, col = colors[i])
}


# Update legend to include AUC values
trial_names_with_auc <- paste(trial_names, ", AUC=", round(AUC_values, 3), sep="")
# Add legend
legend("bottomright", trial_names_with_auc, col = colors, lwd = 2, cex = 0.8)
#-----------------------------------------------------------------------------------------------------
```

7.5 SVM modelling

```
# VSM
# step1: install necessary package
# install the necessary package

install.packages("kernlab")
library(kernlab)
#-----------------------------------------------------------------------------------------------
# 2 Trainning the model
# the data into test and train same as before
# Now split into training (80 percent),
# and test data (20 percent)
ico_size <- floor(0.80 * nrow(ico_new_normalized_num))
# set seed
set.seed(12345)
# split
trainning <- sample(nrow(ico_new_normalized_num), ico_size)
ico_train_svm <- ico_new_normalized_num[trainning, ]
ico_test_svm <- ico_new_normalized_num[-trainning, ]

#class selected only first col = "success"
ico_train_svm_labels <- ico_new_normalized_num[trainning, 1]
ico_test_svm_labels <- ico_new_normalized_num[-trainning, 1]

# begin by training a simple linear SVM
# kernel type:
# 'vanilladot' - linear; 'rbfdot' - rbf; 'ploydot' - ploynomial; 'tanhdot' - Sigmoid
# C=1 is default
ico_classifier_svm <- ksvm(success ~ ., data = ico_train_svm,
                           kernel = "vanilladot")

# Test the model
ico_svm_prediction <- predict(ico_classifier_svm, select(ico_test_svm, -success))
#-----------------------------------------------------------------------------------------------
# Step 3: Evaluating model performance

CrossTable(x = ico_test_svm_labels, y = ico_svm_prediction, prop.chisq=FALSE)

confusionMatrix(ico_test_svm_labels, ico_svm_prediction, positive = "1")
prec = posPredValue(ico_test_svm_labels, ico_svm_prediction, positive = "1") # this is precision
rec =sensitivity(ico_test_svm_labels, ico_svm_prediction, positive = "1") # this is recall

f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1

#-----------------------------------------------------------------------------------------------
```

```
# Improve the model

# Kernel list
kernerlist <- c("rbfdot", "polydot", "tanhdot", "vanilladot")
classifier_names <- c("RBF Kernel", "Polynomial Kernel", "Hyperbolic Tangent Kernel", "Linear Kernel")

# Initialise lists to store models and predictions
svm_models <- list()
predictions_list <- list()

# Train SVM
for (i in seq_along(kernerlist)) {
  # Train SVM model with prob output
  svm_models[[i]] <- ksvm(success ~ ., data = ico_train_svm, kernel = kernerlist[i], type = "C-svc", prob.model = TRUE)
  # Predict test set prob
  probabilities <- predict(svm_models[[i]], select(ico_test_svm, -success), type = "probabilities")[,"1"]
  # Store prob
  predictions_list[[i]] <- probabilities
}

# Numeric for test labels
test_labels_numeric <- as.numeric(ico_test_svm$success == "1")

# Initialise plot for ROC curves
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
SVM with Different Kernels")
abline(a = 0, b = 1, lty = 2, col = "gray")

# Colors for the ROC curves
colors <- rainbow(length(kernerlist))
# Initialise vector to store AUC values
AUC_values <- numeric(length(kernerlist))

# Generate ROC curves and calculate AUC
for (i in seq_along(predictions_list)) {
  # Create prediction object for ROC curve
  pred_object <- prediction(predictions_list[[i]], test_labels_numeric)
  # Calculate the performance of the predictor
  perf <- performance(pred_object, "tpr", "fpr")
  # Calculate AUC
  auc <- performance(pred_object, "auc")
  AUC_values[i] <- auc@y.values[[1]]  # Store AUC value
  # Plot ROC curve
  plot(perf, add = TRUE, col = colors[i])
}

# Update legend to include AUC values
classifier_names_with_auc <- paste(classifier_names, ", AUC=", round(AUC_values, 3), sep="")
# Add legend
legend("bottomright", classifier_names_with_auc, col = colors, lwd = 2, cex = 0.8)
#-------------------------------------------------------------------------------------------------
```

7.6 ANN modelling

```
# ANN
# step1: install necessary package
# install the necessary package
install.packages("neuralnet")
library(neuralnet)
library(Metrics)

#-----------------------------------------------------------------------------------------------
# 2 Trainning the model

# the data into test and train same as before
# Now split into training (80 percent),
# and test data (20 percent)
ico_size <- floor(0.80 * nrow(ico_new_normalized_num))
# set seed
set.seed(12345)
# split
trainning <- sample(nrow(ico_new_normalized_num), ico_size)
ico_train_ann <- ico_new_normalized_num[trainning, ]
ico_test_ann <- ico_new_normalized_num[-trainning, ]

#class selected only first col = "success"
ico_train_ann_labels <- ico_new_normalized_num[trainning, 1]
ico_test_ann_labels <- ico_new_normalized_num[-trainning, 1]

# setseed
set.seed(12345) # to guarantee repeatable results
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 1)

plot(ann_model)

predicted_success <- predict(ann_model, select(ico_test_ann, -success))
predicted_success

# Convert the continuous predictions to binary (0 and 1)
predicted_classes <- ifelse(predicted_success[,1] > 0.5, 0, 1)
predicted_classes <- factor(predicted_classes, levels = c("0", "1"))
#-----------------------------------------------------------------------------------------------
#3. Evaluating model performance
CrossTable(x = ico_test_ann_labels, y = predicted_classes, prop.chisq=FALSE)
confusionMatrix(ico_test_ann_labels, predicted_classes, positive = "1")
prec = posPredValue(ico_test_ann_labels, predicted_classes, positive = "1") # this is precision
rec =sensitivity(ico_test_ann_labels, predicted_classes, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
#-----------------------------------------------------------------------------------------------
# improving the performance
# setseed
set.seed(12345) # to guarantee repeatable results
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 0)
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 1)
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 2)
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 3)
ann_model <- neuralnet(formula = success ~ .,
                       data = ico_train_ann,
                       hidden = 4)

plot(ann_model)
```

```
predicted_success <- predict(ann_model, select(ico_test_ann, -success))
predicted_success

# Convert the continuous predictions to binary (0 and 1)
predicted_classes <- ifelse(predicted_success[,1] > 0.5, 0, 1)
predicted_classes <- factor(predicted_classes, levels = c("0", "1"))
#-------------------------------------------------------------------------------------------------
#3. Evaluating model performance
CrossTable(x = ico_test_ann_labels, y = predicted_classes, prop.chisq=FALSE)
confusionMatrix(ico_test_ann_labels, predicted_classes, positive = "1")
prec = posPredValue(ico_test_ann_labels, predicted_classes, positive = "1") # this is precision
rec =sensitivity(ico_test_ann_labels, predicted_classes, positive = "1") # this is recall
f1 <- (2 * prec * rec) / (prec + rec)
prec
rec
f1
#-------------------------------------------------------------------------------------------------
# ROC
# Define number of hidden neurons
hidden_values <- c(0, 1, 2, 3, 4)
hidden_names <- paste("Hidden layers=", hidden_values)
colors <- c("blue", "green", "red", "purple", "orange")

# Create list to store predictions
predictions_list <- list()

# Training and predict
for (i in 1:length(hidden_values)) {
  set.seed(12345)
  ann_model <- neuralnet(success ~ ., data = ico_train_ann, hidden = hidden_values[i], stepmax = 1e+06)
  # collect prob
  predictions_list[[i]] <- predict(ann_model, ico_test_ann[, -1], type = "raw")[, 2]
}

# Create plot
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
ANN 1 Layers with Different Hidden Neuron ")
abline(a = 0, b = 1, lty = 2, col = "gray")

# Create vector to store AUC values
AUC_values <- numeric(length(predictions_list))

# Loop to calculate and plot each ROC curve
for (i in 1:length(predictions_list)) {
  # Convert predictions to numeric
  test_pred_numeric <- predictions_list[[i]]
  test_labels_numeric <- as.numeric(as.character(ico_test_ann_labels))
  # Create prediction object for ROC curve
  pred_object <- prediction(test_pred_numeric, test_labels_numeric)
  # Calculate the performance of the predictor
  perf <- performance(pred_object, "tpr", "fpr")
  # Calculate AUC
  auc <- performance(pred_object, "auc")
  AUC_values[i] <- auc@y.values[[1]]  # Store AUC value
  # Plot ROC curve
  plot(perf, add = TRUE, col = colors[i])
}

# Prepare legend text including AUC values
hidden_names_with_auc <- paste(hidden_names, ", AUC=", round(AUC_values, 3), sep="")
# Add legend
legend("bottomright", hidden_names_with_auc, col = colors, lwd = 2, cex = 0.8)
#-------------------------------------------------------------------------------------------------
```

7.7 Compared ROC plot.

```
# ROC
# all model to compared
ico_test_pred <- knn(train = ico_train, test = ico_test, cl = ico_train_labels, k=49)
ico_classifier_nv1 <- naiveBayes(ico_train_nv, ico_train_nv_labels, laplace = 1)
dt_model <- C5.0(x = select(ico_train_dt, -success), y = ico_train_dt$success, trials = 5)
svm_models <- ksvm(success ~ ., data = ico_train_svm, kernel = "vanilladot", type = "C-svc", prob.model = TRUE)
ann_model <- neuralnet(success ~ ., data = ico_train_ann, hidden = 3, stepmax = 1e+06)

# Predictions
# KNN
test_pred_numeric <- as.numeric(as.character(ico_test_pred ))
test_labels_numeric <- as.numeric(as.character(ico_test_labels))

ico_test_pred_knn <- prediction(test_pred_numeric, test_labels_numeric)
# Naive Bayes
ico_test_pred_nb <- predict(ico_classifier_nv1, select(ico_test_nv, -success), type = "raw")[, 2]
# Decision Tree
ico_test_pred_dt <- predict(dt_model, ico_test_dt, type = "prob")[, "1"]
# SVM
ico_test_pred_svm <- predict(svm_models, select(ico_test_svm, -success), type = "probabilities")[, "1"]
# ANN
ico_test_pred_ann <- predict(ann_model, ico_test_ann[, -1], type = "raw")[, 2]

# Ensure labels are in numeric format
ico_test_labels_numeric <- as.numeric(as.character(ico_test_labels))
# Prepare the ROC plot
plot(0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate", xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for
Multiple Models")
abline(a = 0, b = 1, lty = 2, col = "gray")
# Colors for plots
colors <- c("blue", "green", "red", "purple", "orange")
# Calculate and plot ROC for each model
model_predictions <- list(ico_test_pred_knn, ico_test_pred_nb, ico_test_pred_dt, ico_test_pred_svm, ico_test_pred_ann)
model_names <- c("KNN", "Naive Bayes", "Decision Tree", "SVM", "ANN")

for (i in seq_along(model_predictions)) {
  pred <- prediction(model_predictions[[i]], ico_test_labels_numeric)
  perf <- performance(pred, "tpr", "fpr")
  plot(perf, add = TRUE, col = colors[i])
}

# Add legend
legend("bottomright", legend=model_names, col=colors, lwd = 2, cex = 0.8)
```